Advance Algorithms (CS354)

Project Report

Karger-Stein min cut algorithm

Team Members:

Male Nikhitha Reddy **21CSB0B34**

Naga Sai Charitha **21CSB0B45**

Tattukolla Charishma **21CSB0B59**

Under Prof. Manish Kumar Bajpai

Introduction :

Problem Definition:

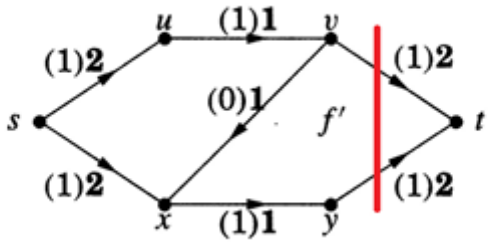
Given a graph with n vertices and m (possibly weighted) edges, we wish to partition the vertices into two nonempty sets to minimize the number or total weight of edges crossing between them.

The problem has two variants:

- In the **s-t min-cut problem**, we require that the two specified vertices s and t be on opposite sides of the cut.
- In what will be called the min-cut problem, or for emphasis the **global min-cut problem**, there is no specific two vertices.

Max- flow Min-cut Theorem (Ford and Fulkerson,1956):

In every network, the maximum value of a feasible flow equals the minimum capacity of a source/sink cut.



The minimum cut problem has many Applications like:

- The problem of determining the connectivity of a network arises frequently in issues of network design and network reliability.
- In information retrieval, minimum cuts have been used to identify clusters of topically related documents in hypertext systems.

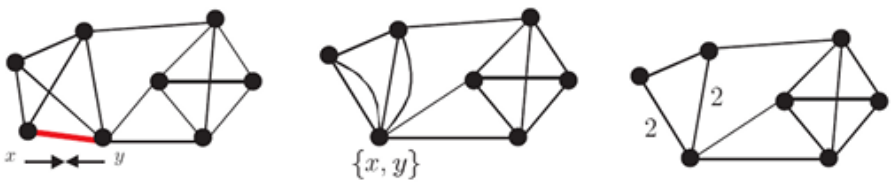
- Minimum cut problems also play an important role in large-scale combinatorial optimization
- Minimum cut problems arise in the design of compilers for parallel languages.

Karger's Algorithm:

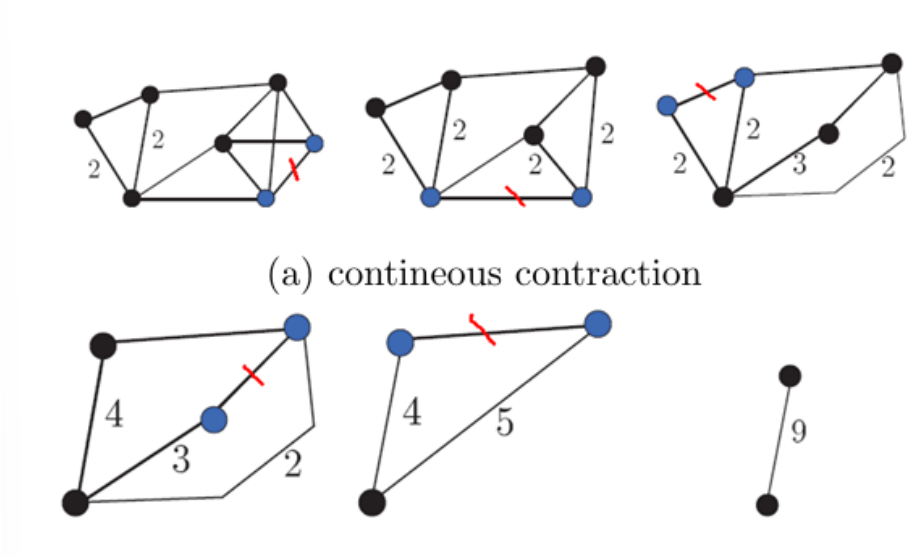
Karger's algorithm is a randomized algorithm to find the minimum cut of a graph. It works by iteratively contracting random edges of the graph until only two nodes remain. The remaining two nodes represent the two partitions of the minimum cut.

Contraction Algorithm:

The fundamental idea of Karger's algorithm is a form of edge contraction. In a graph G, contraction of edge e with endpoints u, v is the replacement of u and v with single vertex whose incident edges are the edges other than e that were incident to u or v. the resulting graph, denoted as G/e, has one less edge that G.



(a) Contract edge xy (b) merge into one vertex (c) after contraction



The Pseudocode is presented as below:

```
procedure MinCut (G)
while |V|>2
        choose e in E(G) uniformly at random
        G = G/e
  return the only cut in G
```

The algorithm always output a cut, and the cut is not smaller than the minimum cut. The algorithm runs in O(n^2) time.

Algorithm Analysis:

MinCut algorithm outputs the min cut in probability $\mathbb{P} \geq \frac{2}{n(n-1)}$

The probability that repeat MinCut algorithm $T = \binom{n}{2} \log n$ times fails to return the minimum cut is $< \frac{1}{n}$

We can compute the minimum cut in O(n^4) time with constant probability to get a correct result. In O(n^4log n) time the minimum cut is returned with high probability.

Karger-Stein Algorithm:

An extension of Karger's algorithm due to David Karger and Clifford Stein achieved an order of magnitude improvement. They also call it Recursive Contraction Algorithm. The basic idea is to perform the contraction procedure until the graph reaches $1+n/\sqrt{2}$ vertices.

The Karger-Stein algorithm has an expected runtime of approximately $O(n^2 \log^3 n)$, making it more efficient than Karger's algorithm for large graphs.

Recursive Contraction Algorithm:

As the graph get smaller, the probability to make a bad choice increases. So, run the algorithm more times when the graph is smaller.

Here is the Pseudo code:

```
procedure MinCut(G)
while |V|>2
      choose e in E(G) uniformly at random
      G = G/e
  return the only cut in G
```

Karger and Stein provide a new algorithm with **recursive process**:

Here is the Pseudocode for recursive contraction:

```
procedure FastMinCut (G)
if |V| < 6
    MinCut(G,2)
else
   t = 1 + |V|/sqrt(2)
    G1 = MinCut(G,t)
    G2 = MinCut(G,t)
return min (FastMinCut(G1), FastMinCut(G2))
```

Algorithm Analysis:

The running time of FastMinCut (G) is O(n^2 log n) , where n=|V(G)| .

$$T(n)=O(n^2)+2T(\frac{n}{\sqrt{2}})$$

Well, we perform two calls to Contract (G ,t) which takes O(n^2) time. And then we perform two recursive calls, on the resulting graphs. We have: The solution to this recurrence is O(n^2 log n) as one can easily verify.

Running FastMinCut finds the minimum cut with probability larger that $\frac{2\log 2}{\log n}$ *,which can be notated as* $\Omega(1/\log n)$

Implementation:

To specific the Min-Cut problem, here is an example with 200 vertices. It provides codes with Python:

By running below algorithm n^2 log n times, thus we can get the min-cut with high probability.

For the simple **MinCut algorithm**, the codes are implemented as follows:

```
def MinCut(graph, t):
    while len(graph) > t:
        start = random.choice(graph.keys())
        finish = random.choice(graph[start])

    # # Adding the edges from the absorbed node:
        for edge in graph[finish]:
            if edge != start:  # this stops us from making a self-loop
                graph[start].append(edge)

    # # Deleting the references to the absorbed node and changing them to
    #the source node:
        for edge1 in graph[finish]:
            graph[edge1].remove(finish)
            if edge1 != start:
                # this stops us from re-adding all the edges in start.
                graph[edge1].append(start)
        del graph[finish]

    # # Calculating and recording the mincut
    mincut = len(graph[graph.keys()[0]])
    cuts.append(mincut)
    return graph
```

Output:

```
D:\AAproject\algo>python MinCut.py
Karger-Stein:
Total edges:      2517.0
Total vertices:   200
Runing times:     58
Mincut is         17
```

```
filename = "KargerMinCut.txt"
  file1 = open(filename)
  graph = {}
  cuts = []
  edge_num = 0
  edge_list = []
  for line in file1:
      node = int(line.split()[0])
      edges = []
      for edge in line.split()[1:]:
          edges.append(int(edge))
      graph[node] = edges
      edge_num = edge_num + len(edges)
      edge_list.append(len(edges))
  file1.close()
  count = len(graph) * len(graph) * int(math.log(len(graph)))
  # running times
  while i++ < count:
      graph1 = copy.deepcopy(graph)
      g = MinCut(graph1,2)
```

For **Karger Stein algorithm** , we implemented as recursive way, code follows:

```
def FastMinCut(graph):
    if len(graph) < 6:
        return BaseMinCut(graph, 2)
    else:
        t = 1 + int(len(graph) / math.sqrt(2))
        graph_1 = MinCut(graph, t)
        graph_2 = MinCut(graph, t)
        return min(FastMinCut(graph_1), FastMinCut(graph_2))

count = int(math.log(len(graph))) * int(math.log(len(graph)))
# running times
  while i++ < count:
      graph1 = copy.deepcopy(graph)
      g = FastMinCut(graph1)
```

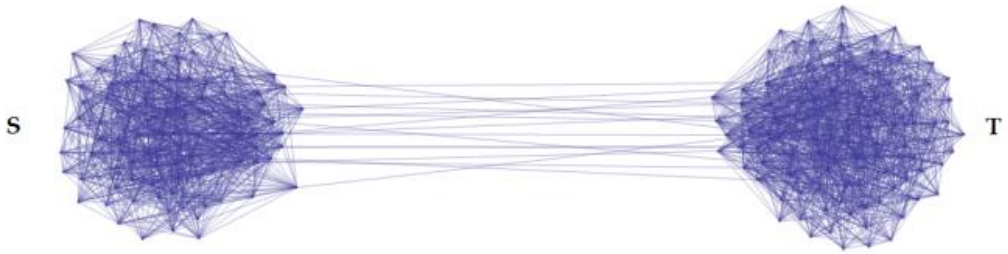


Figure 6: Partition into two partite with MinCut=17

Output:

```
D:\AAproject\algo>python MinCut.py
Kargers:
Total edges:      2517.0
Total vertices:   200
Runing times:     305754
Mincut is         17
```

It outputs the same result, while consumes much less time.

Conclusion:

Let's look at the ***comparison*** of the min-cut problem:

From the table below, we can get that Karger-Stein Algorithm gets quite close to the minimum order O(n^2).

- *s-t min-cut problem*

Bound	unweighted graph	weighted graph
Directed	$cm \log \frac{n^2}{m}$ [5]	$mn \log \frac{n^2}{m}$ [4]
Undirected	$c^2 n \log \frac{n}{c}$ [5]	$mn + n^2 \log n$ [6]

- *global min-cut problem*

Bound	unweighted graph	weighted graph
Directed	$\mathcal{P} - complete$	$\mathcal{P} - complete$
Undirected	$O(n^2 \log^3 n)$ - Karger-Stein [8]	$O(n^2 \log^3 n)$ - Karger-Stein [8]

Then take a look at comparision of Karger's algorithm and Karger-Stein algorithm.

Bound	Karger algorithm	Karger-Stein algorithm
Probability	$O(1/n^2)$	$O(1/\log n)$
Cost	$O(n^2)$	$O(n^2 \log n)$
Running times	$\binom{n}{2} \log n$	$\log^2 n$
Totol Order	$O(n^4 \log n)$	$O(n^2 \log^3 n)$

LIMITATIONS:

- Like Karger's algorithm, the Karger-Stein algorithm provides an approximate solution to the minimum cut problem. While it reduces variance and improves accuracy compared to Karger's algorithm, it is not guaranteed to find the exact minimum cut in all cases.
- Like many randomized algorithms, the Karger-Stein algorithm may converge to a local minimum

IMPROVISATIONS:

While the Karger-Stein algorithm quite efficient compared to Karger's algorithm, further reduction in running times can be explored. This would involve optimizing the recursive contraction process and possibly exploring parallel algorithms.